REMARKS

This Amendment is filed in response to the FINAL Office Action mailed on January 26, 2009, and in response to a telephone interview held with Examiner Shawn Eland on or about January 16, 2009, and with the Request for Continued Examination (RCE) filed on even date herewith. All objections and rejections are respectfully traversed.

Claims 1-11, 13-19, 22-26, and 28-62 are pending.

Claims 1, 11, 22, 32 and 38 were amended to better claim the invention.

Claim 50-62 were added to better claim the invention.

Claims 21-22 were cancelled without prejudice.

Request for Examiner Interview

The Applicant respectfully requests a telephonic interview with the Examiner after the Examiner has had an opportunity to consider this Amendment, but before issuance of the next Office Action. The Applicant's undersigned attorney may be reached at 617-951-2500.

Interview Summary

Applicant would like to take this opportunity to thank Examiner Shawn Eland for his courtesy and helpfulness in the telephone interview held on 16 January 2009. Examiner

Eland suggested that the claims be amended to indicate that write operations to the storage system continue while a consistency point (CP) is being written to persistent storage. The claims have been so amended.

Rejection of claims 11, 13-19, and 49 under 35 U.S.C. 103(a)

At Paragraph 5 (Page 3) of the Office Action, claims 11, 13-19, and 49 were rejected under 35 USC 103(a) in view of Hitz US Patent 5,819,292 (hereinafter Hitz) in view of Bush et al US Patent 5,790,778 (hereinafter Bush).

Applicant has reviewed Hitz for possible exclusion from a 103(a) rejection under 103(c), as both Hitz and the present application are owned by Network Appliance, Inc. However, Hitz issued on 6 October 1998, and the present Applicant for United States Patent was filed on 11 October 2003, approximately 5 years after Hitz issued. Accordingly, Hitz qualifies as prior art under 35 U.S.C. 102 (a), and section 103 (c) does not apply.

Applicant respectfully urges that the present Application is definitely an improvement over the disclosures of Hitz, and is made by the owner of Hitz, and is patentably distinct from all cited art..

Applicant's claimed invention, as set forth by representative claim 11, comprises in part:

11. A method for detecting leaked buffer writes between a first consistency point and a second consistency point, comprising:

initiating writing a first consistency point (CP) at a first time, a CP being a wholly consistent and up-to-date version of an old data of the file system which is written to persistent storage, the old data received by the storage system since an earlier CP was written to persistent storage, and the old data stored in an old buffer in memory of the storage system;

determining a first consistency point number assigned to the first CP, the first CP number to identify the old buffers as holding data to be written to persistent storage during the first CP;

selecting an old data buffer;

determining if one or more uniquely identifying numbers (hereinafter magic numbers) are within the old data buffer, the magic numbers to uniquely identify the old data buffer as having a labeled buffer check control structure and to indicate that the old data buffer needs to be checked for leakage;

reading an identifying consistency point number from the labeled buffer check control structure;

receiving a write operation to add new data to the storage system, the new data to be written to a second CP at a later second time, the new data written to a new data buffer in memory of the storage system;

comparing a buffer consistency point number read from the old buffer with the first consistency point number, and if the buffer consistency point number agrees with the first consistency point number, continuing to write the first consistency point, and continuing to write the new data to the new data buffer, and

> ***in the event that the buffer consistency point number disagrees with the first consistency point number, signaling an administrator that buffer leakage between consistency points has occurred, and halting writing of the first consistency point, and halting write operations to add new data to the storage system***.

Hitz is incorporated by reference in Applicant's Specification at Page 5 lines 12-15, as a description of Consistency Points in time (CP). At Col 12 beginning at line 40, Hitz discloses an algorithm for generating a Consistency Point. Hitz stops writes by clients while a Consistency Point is being written.

Problem Solved by the Present Invention

The problem solved by the present invention is to maintain receiving write commands from clients while writing a consistency point (CP), and the problem is set out in Applicant's Specification in the BACKGROUND section as follows,

"A file system typically stores incoming (write) data to be written to disk in one or more data buffers associated with the file or other data container, e.g. a virtual disk (vdisk), etc. In storage appliances that utilize current and next consistency points, it is possible to have data associated with more than one consistency point in a data buffer at a given time. Information, such as pointers referencing the data buffers associated with specific CPs, stored in the buffer control structure are not guaranteed to correctly identify the CP associated with a specific write operation as these pointers may become corrupted. For example, the value of a pointer may be inadvertently altered to that of another pointer such that the data from one data buffer may "leak" from its proper CP to another CP." (Specification, Page 6 line 26 – Page 7 line 4)

Hitz states, At Col 12 lines 44-48:

"Only when those writes are complete are any writes from other inodes allowed to reach disk. Further, during the time dirty writes are occurring, no new modifications can be made to inodes that have their consistency point flag set."
Hitz (Col 12 lines 44-48)

Accordingly, Hitz stops writes by clients while a Consistency Point is being written, and is illustrative of the problem being solved by the present invention, as set out hereinabove.

Bush discloses a first computer program which checks a second computer program for errors. An error which is checked for is "memory leakage" and "resource leakage". Bush defines leakage at his Col. 26 line 38 – 49 as follows:

"Action 1212 performs leak detection processing. Action 1212 loops through all of modeled memory and scans the information about memory allocation accumulated during analysis of the current path. Action 1212 identifies any chunk of memory that will be leaked when the current function exits. A piece of memory is **leaked** when it is allocated, but it will not be pointed to by any symbol after the function exits. Action 1212 also detects leaked resources. A detailed explanation of the processing performed by action 1212 is given below under the heading "Leak Detection". Processing transfers from action 1212 to reset pragma.sub.-- options action 1214 (hereinafter "action 1214")."

Applicant respectfully urges that neither Hitz not Bush discloses Applicant's claimed putting a write operation into the proper consistency point by using Applicant's claimed novel: ***initiating writing a first consistency point (CP) at a first time . . . determining a first consistency point number assigned to the first CP . . . determining if one or more uniquely identifying numbers (hereinafter magic numbers) are within the old data buffer . . . reading an identifying consistency point number from the labeled buffer check control structure . . . receiving a write operation to add new data to the storage system, the new data to be written to a second CP at a later second time . . . comparing a buffer consistency point number read from the old buffer with the first consistency point number, and if the buffer consistency point number agrees with the first consistency point number, continuing to write the first consistency point, and continuing to write the new data to the new data buffer, and in the event that the buffer consistency point number disagrees with the first consistency point number, signaling an administrator that buffer leakage***

between consistency points has occurred, and halting writing of the first consistency point, and halting write operations to add new data to the storage system.

That is, Applicant respectfully urges that neither Hitz nor Bush disclose Applicant's claimed ***comparing a buffer consistency point number read from the old buffer with the first consistency point number, and if the buffer consistency point number agrees with the first consistency point number, continuing to write the first consistency point, and continuing to write the new data to the new data buffer.***

Further, Applicant respectfully urges that neither Hitz nor Bush disclose Applicant's claimed ***in the event that the buffer consistency point number disagrees with the first consistency point number, signaling an administrator that buffer leakage between consistency points has occurred, and halting writing of the first consistency point, and halting write operations to add new data to the storage system***.

Even further, neither Hitz nor Bush discloses Applicant's claimed use of ***magic numbers*** to identify the proper consistency point for a write, Applicant respectfully urges that neither Hitz nor Bush, taken singly or in combination, can legally render Applicant's claimed invention unpatentable under 35 U.S.C. 102(a).

The use of ***magic numbers*** to identify elements in computer programming is well known to those skilled in the art of computer programming, especially in programming in a Unix environment. A passage from Wikepedia concerning the use of ***magic numbers*** in computer programming follows:

Magic number (programming)

From Wikepedia, the free encyclopedia

Jump to: navigation, search
For other uses of the term, see magic number.

In computer programming, the term **magic number** has multiple meanings. It could refer to one or more of the following:

- a constant used to identify a file format or protocol;
- an unnamed and/or ill-documented numerical constant; or
- distinctive debug values or GUIDs, etc.

Contents

[hide]

- 7 References

[edit] Format indicator

[edit] Magic number origin

The type of magic number was initially found in early Seventh Edition source code of the Unix operating system and, although it has lost its original meaning, the term **magic number** has become part of computer industry lexicon.

When Unix was ported to one of the first DEC PDP-11/20s it did not have memory protection and, therefore, early versions of Unix used the relocatable memory reference model.[1] Thus, pre-Sixth Edition Unix versions read an executable file into memory and jumped to the first low memory address of the program, relative address zero. With the development of paged versions of Unix, a header was created to describe the executable image components. Also, a branch instruction was inserted as the first word of the header to skip the header and start the program. In this way a program could be run in the older relocatable memory reference (regular) mode or in paged mode. As more executable formats were developed, new constants were added by incrementing the branch offset.[2]

In the Sixth Edition source code of the Unix program loader, the exec() function read the executable (binary) image from the file system. The first 8 bytes of the file was a header containing the sizes of the program (text) and initialized (global) data areas. Also, the first 16-bit word of the header was compared to two constants to determine if the executable image contained relocatable memory references (normal), the newly implemented paged read-only executable image, or the separated instruction and data paged image.[3] There was no mention of the dual role of the header constant, but the high order byte of the constant was, in fact, the operation code for the PDP-11 branch instruction (octal 000407 or hex 0107). Adding seven to the program counter showed that if this constant was executed, it would branch the Unix exec() service over the executable image eight byte header and start the program.

Since the Sixth and Seventh Editions of Unix employed paging code, the dual role of the header constant was hidden. That is, the exec() service read the executable file header (meta) data into a kernel space buffer, but read the executable image into user space, thereby not using the constant's branching feature. Magic number creation was implemented in the Unix linker and loader and magic number branching was probably still used in the suite of stand-alone diagnostic programs that came with the Sixth and Seventh Editions. Thus, the header constant did provide an illusion and met the criteria for magic.

In Version Seven Unix, the header constant was not tested directly, but assigned to a variable labeled **ux_mag**[4] and subsequently referred to as the **magic number**. Given that there were approximately 10,000 lines of code and many constants employed in these early Unix versions, this indeed was a curious name for a constant, almost as curious as the [1] comment used in the context switching section of the Version Six program manager. Probably because

of its uniqueness, the term **magic number** came to mean executable format type, then expanded to mean file system type, and expanded again to mean any strongly typed file.

Accordingly, Applicant respectfully urges that the concept of ***magic numbers*** is well known in the computer arts.

However, Applicant respectfully urges that it is completely novel to use magic numbers to identify a consistency point into which a data buffer should be written, as claimed by Applicant as: ***"determining if one or more uniquely identifying numbers (hereinafter magic numbers) are within the old data buffer, the magic numbers to uniquely identify the old data buffer as having a labeled buffer check control structure and to indicate that the old data buffer needs to be checked for leakage . . . comparing a buffer consistency point number read from the old buffer with the first consistency point number, and if the buffer consistency point number agrees with the first consistency point number, continuing to write the first consistency point, and continuing to write the new data to the new data buffer"***.

At pages 3-4 of the Office Action it is urged that Hitz teaches, at Col 12 lines 11-48, a "buffer check control structure", and a "consistency point number stored in the buffer check control structure". Hitz, at Col. 12 lines 11-48 states:

"Referring to FIG. 16, a new consistency point is written by first flushing all file system blocks to new locations on disk (including the blocks in meta-data files such as the inode file 1620, blkmap file 1630, and inomap file 1640). A new root inode 1610B and

1612B for the file system 1670 is then written to disk. With this method for atomically updating a file system, the on-disk file system is never inconsistent. The on-disk file system 1670 reflects an old consistency point up until the root inode 1610B and 1612B is written. Immediately after the root inode 1610B and 1612B is written to disk, the file system 1670 reflects a new consistency point. Data structures of the file system 1670 can be updated in any order, and there are no ordering constraints on disk writes except the one requirement that all blocks in the file system 1670 must be written to disk before the root inode 1610B and 1612B is updated.

To convert to a new consistency point, the root inode 1610B and 1612B must be updated reliably and atomically. WAFL does this by keeping two identical copies of the fsinfo structure 1610 and 1612 containing the root inode 1610B and 1612B. During updating of the root inode 1610B and 1612B, a first copy of the fsinfo structure 1610 is written to disk, and then the second copy of the fsinfo structure 1612 is written. A checksum 1610C and 1612C in the fsinfo structure 1610 and 1612, respectively, is used to detect the occurrence of a system crash that corrupts one of the copies of the fsinfo structure 1610 or 1612, each containing a copy of the root mode, as it is being written to disk. Normally, the two fsinfo structures 1610 and 1612 are identical.

Algorithm for Generating a Consistency Point

FIG. 5 is a diagram illustrating the method of producing a consistency point. In step 510, all "dirty" inodes (inodes that point to new blocks containing modified data) in the system are marked as being in the consistency point. Their contents, and only their contents, are written to disk. Only when those writes are complete are any writes from other inodes allowed to reach disk. Further, during the time dirty writes are occurring, no new modifications can be made to inodes that have their consistency point flag set."

Again, Applicant respectfully points out that Hitz stops writes by clients while a Consistency Point is being written, as pointed out hereinabove, at the quoted Col. 12 lines 44-48 which state: "Further, during the time dirty writes are occurring, no new modifications can be made to inodes that have their consistency point flag set."

Accordingly, a careful reading of Hitz indicates that Hitz has no reference to Applicant's claimed novel a "buffer check control structure", and a "consistency point number stored in the buffer check control structure".

The invention allows write operations to flow to Applicant's claimed system while a Consistency Point is being written, therefore solving the stated problem.

At Page 4 of the Office Action, last paragraph, it is urged that:

"Bush however teaches magic numbers (see col. 28 line 53 through col. 29 line 28), and even though he does not explicitly teach the magic number as comprising either a 64-bit number, two 32-bit number, nor the consistency point number as comprising a 32-bit number, such limitations are merely a design choice and would have been obvious in the system of Bush. These limitations fail to define a patentably distinct invention over Bush since both the invention as a whole and that of Bush are directed to storing a magic number used to uniquely identify the data block; and storing a consistency number, used to track certain points in time the system maintained a consistent state."

At the cited portion of Bush, Bush states:

"Action 1504 creates a model for one or more contiguous memory locations. A memory location is the smallest unit of memory that can be explicitly and uniquely specified by means of an address. Typically, computer memory is byte addressable, and thus, a location is one byte. Action 1504 models memory using a chunk 1700. Chunk 1700 is shown in FIG. 17. Chunk 1700 includes fields: "freed flag" 1702, "reachable flag" 1704, "lost flag" 1706, "memory type" 1708, "chunk number" 1710, "origin context structure pointer" 1712, "stored value pointer" 1714 and "original stored value pointer" 1716.

Flag "freed flag" is true when the memory locations modeled by chunk 1700 have been freed. Flag "reachable flag" 1702 is used by leak detection processing to determine if the memory location is reachable. Flag "lost flag" 1706 is true when it can not be determined if the memory modeled is freed or leaked. With lost memory, it is possible that nothing will point to the memory after the function exits, but just because there is no record of a pointer to the memory does not mean that such a pointer does not exist. For example, memory can be allocated and then passed to a routine which is modeled by the missing model. Analysis engine 308 can not ascertain what happened to the allocated memory passed into the routine. Thus, the memory is marked as "lost". Field "memory type" 1708 holds the same information as field "memory type" 1606 described above. Field "chunk number" 1710 is a unique identifier for chunk 1700. Field "origin context structure pointer" 1712 points to the origin context structure 1600 created in action 1502. Field "stored value pointer" 1714 points to the current value in the modeled memory location. Field "original stored value pointer" 1716 points to the original value in the modeled memory location.

First, action 1504 iterates through the chunk table looking at chunks 1700 to determine if a chunk 1700 can be reused. If action 1504 can not reuse any chunks 1700 then it must create a new chunk 1700. A pointer to the new chunk 1700 is put into the chunk table. Chunk number 1710 is assigned a number that uniquely identifies new chunk 1700. Flags "freed flag" 1702, "reachable flag" 1704 and "lost flag" 1706 are initialized to false. Field "memory type" 1708 is set to equal "memory type" 1606 set in action 1502. Field "origin context structure pointer" 1712 is set to point to the origin context structure 1600 built in action 1502. Processing then transfers to model values action 1506 (hereinafter "action 1506") to create the stored value set."

A careful reading of Bush at col. 28 line 53 through col. 29 line 28 indicates that Bush's "chunk number" is the structure being compared with Applicant's claimed "magic numbers".

However Bush's "chunk numbers" are a sequential numbering of his "memory chunks". Applicant's "magic numbers" are arbitrary numbers chosen to identify a data buffer as having a particular property. In Applicant's claimed invention, the claimed "magic numbers" identify a data buffer as having ***the magic numbers to uniquely identify the old data buffer as having a labeled buffer check control structure and to indicate that the old data buffer needs to be checked for leakage***.

Referring to the Wikipedia description of magic numbers, it is clear that a "magic number" is not a simple sequential numbering of objects, as Bush's "chunk numbers" are. Therefore, Applicant respectfully urges that Bush's sequential "chunk numbers" are totally different from Applicant's claimed "magic numbers", and that Bush's chunk numbers do not anticipate and do not render unpatentable Applicant's claimed novel "magic numbers".

Accordingly, Applicant respectfully urges that Hitz and Bush, taken either singly or in combination are legally incapable of rendering Applicant's claimed invention unpatentable under 35 U.S.C. 103(a) because of the absence from each of Applicant's claimed novel

initiating writing a first consistency point (CP) at a first time . . .

determining a first consistency point number assigned to the first CP . . . determining if one or more uniquely identifying numbers (hereinafter magic numbers) are within the old data buffer . . . reading an identifying consistency point number from the labeled buffer check control structure . . . receiving a write operation to add new data to the storage system, the new data to be written to a second CP at a later second time . . .

comparing a buffer consistency point number read from the old buffer with the first consistency point number, and if the buffer consistency point number agrees with the first consistency point number, continuing to write the first consistency point, and continuing to write the new data to the new data buffer, and

> ***in the event that the buffer consistency point number disagrees with the first consistency point number, signaling an administrator that buffer leakage between consistency points has occurred, and halting writing of the first consistency point, and halting write operations to add new data to the storage system.***

Rejection of claims 1-10, 20-26, and 28-48 under 35 U.S.C. 103(a)

At Paragraph 6 of the Office Action, Claims 1-10. 20-26, and 28-48 were rejected under 35 U.S.C. 102(a) as being unpatentable over Hitz, Marion et al. U. S. Patent Publication 2003 / 0163661 A1, and further in view of Bush.

Applicant's claimed invention, as set out in representative claim 1, comprises in part:

"1. A method for operating a storage system,

comprising:

initiating writing a first consistency point (CP) at a first time, a CP being a wholly consistent and up-to-date version of an old data of the file system which is written to persistent storage, the old data received by the storage system since an earlier CP was written to persistent storage, and the old data stored in an old buffers in memory of the storage system;

determining a first consistency point number assigned to the first CP, the first CP number to identify the old buffers as holding data to be written to persistent storage during the first CP;

receiving a write operation to add new data to the storage system, the write operation identifying a file for the write operation to store the new data, the new data written to a new buffer in memory of the storage system;

continuing to receive write operations for the new data during writing the first CP, the new data to be written to a second CP at a later second time, the new data written to a new data buffers in memory of the storage system;

determining that a volume storing the file has buffer leakage detection activated;

while writing the new data to the new data buffers, and in response to determining that the volume has buffer leakage detection activated, writing a buffer check control structure to the new data buffer, the buffer check control structure including one or more uniquely identifying numbers referred to as magic numbers, and an identifying consistency point number, the magic numbers to uniquely identify the new data buffer as a labeled buffer check control structure and to indicate that the data buffer needs to be checked for leakage; and

comparing a buffer consistency point number read from an old buffer with the first consistency point number, the magic numbers to identify the buffer check control structure containing the buffer consistency point number, and if the buffer consistency point number agrees with the first consistency point number, continuing to write the first consistency point, and continuing to write the new data to the new data buffers, and

in the event that the buffer consistency point number disagrees with the first consistency point number, signaling an administrator that buffer leakage between consistency points has occurred, and halting writing of the first consistency point, and halting write operations to add new data to the storage system."

Marion discloses a system for detecting memory leaks caused by allocating a region of memory and then failing to de-allocate that region. Marion sets a flag to indicate that the region was allocated, and then later checks to see if the flag is still set. If the flag is still set, Marion then de-allocates the memory region.

Accordingly, Applicant respectfully urges that also Marion has no disclosure of Applicant's claimed novel:

initiating writing a first consistency point (CP) at a first time, . . .

determining a first consistency point number assigned to the first CP . . .

receiving a write operation to add new data to the storage system . . . comparing a buffer consistency point number read from an old buffer with the first consistency point number, the magic numbers to identify the buffer check control structure containing the buffer consistency point number, and if the buffer consistency point number agrees with the first consistency point number, continuing to write the first consistency point, and continuing to write the new data to the new data buffers, and

in the event that the buffer consistency point number disagrees with the first consistency point number, signaling an administrator that buffer leakage between consistency points has occurred, and halting writing of the first consistency point, and halting write operations to add new data to the storage system."

Applicant respectfully urges that since Hitz, Bush, and Marion, either singly or in any combination, do not discloses Applicant's claimed ***comparing a buffer consistency point number read from an old buffer with the first consistency point number, the magic numbers to identify the buffer check control structure containing the buffer consistency point number, and if the buffer consistency point number agrees with the first consistency point number, continuing to write the first consistency point, and continuing to write the new data to the new data buffers, and***

in the event that the buffer consistency point number disagrees with the first consistency point number, signaling an administrator that buffer leakage between consistency points has occurred, and halting writing of the first consistency point, and halting write operations to add new data to the storage system, Applicant respectfully urges that neither Hitz nor Bush, nor Marion, taken singly or in combination, can legally render Applicant's claimed invention unpatentable under 35 U.S.C. 102(a).

SUPPORT IN SPECIFICATION FOR CLAIM ELEMENTS

All elements set out in Amendments to the claims, and in the new claims, are fully supported in the Specification.

Claim 1 will be used as a guide to various elements of the claim, and to their support in the Specification:

1. A method for operating a storage system,
comprising:

initiating writing a first consistency point (CP) at a first time, a CP being a wholly consistent and up-to-date version of an old data of the file system which is written to persistent storage, the old data received by the storage system since an earlier CP was written to persistent storage, and the old data stored in an old buffers in memory of the storage system;

determining a first consistency point number assigned to the first CP, the first CP number to identify the old buffers as holding data to be written to persistent storage during the first CP;

receiving a write operation to add new data to the storage system, the write operation identifying a file for the write operation to store the new data, the new data written to a new buffer in memory of the storage system;

continuing to receive write operations for the new data during writing the first CP, the new data to be written to a second CP at a later second time, the new data written to a new data buffers in memory of the storage system;

determining that a volume storing the file has buffer leakage detection activated;

while writing the new data to the new data buffers, and in response to determining that the volume has buffer leakage detection activated, writing a buffer check control structure to the new data buffer, the buffer check control structure including one or more uniquely identifying numbers referred to as magic numbers, and an identifying consistency point number, the magic numbers to uniquely identify the new data buffer as a labeled buffer check control structure and to indicate that the data buffer needs to be checked for leakage; and

comparing a buffer consistency point number read from an old buffer with the first consistency point number, the magic numbers to identify the buffer check control structure containing the buffer consistency point number, and if the buffer consistency point number agrees with the first consistency point number, continuing to write the first consistency point, and continuing to write the new data to the new data buffers, and

> in the event that the buffer consistency point number disagrees with the first consistency point number, signaling an administrator that buffer leakage between consistency points has occurred, and halting writing of the first consistency point, and halting write operations to add new data to the storage system .

The clause:

initiating writing a first consistency point (CP) at a first time, a CP being a wholly consistent and up-to-date version of an old data of the file system which is written to persistent storage, the old data received by the storage system since an earlier CP was written to persistent storage, and the old data stored in an old buffers in memory of the storage system;

is fully supported in the Specification at: Page 5 lines 3-11.

The clause:

determining a first consistency point number assigned to the first CP, the first CP number to identify the old buffers as holding data to be written to persistent storage during the first CP;

is fully supported in the Specification at: Page 7 line 27 – Page 8 line 4.

The clause:

receiving a write operation to add new data to the storage system, the write operation identifying a file for the write operation to store the new data, the new data written to a new buffer in memory of the storage system;

is fully supported in the Specification at: Page 5 lines 7-11; Fig. 9; Lines Page 24 lines 2-22.

The clause:

continuing to receive write operations for the new data during writing the first CP, the new data to be written to a second CP at a later second time, the new data written to a new data buffers in memory of the storage system;

is fully supported in the Specification at: Page 7 line 17 – Page 8 line 4; Fig. 5; Page 21 lines 6-17;

The clause:
determining that a volume storing the file has buffer leakage detection activated;

is fully supported in the Specification at: Fig. 8 step 815; Page 23 line 1-30.

The clause:

while writing the new data to the new data buffers, and in response to determining that the volume has buffer leakage detection activated, writing a buffer check control structure to the new data buffer, the buffer check control structure including one or more uniquely identifying numbers referred to as magic numbers, and an identifying consistency point number, the magic numbers to uniquely identify the new data buffer as a labeled buffer check control structure and to indicate that the data buffer needs to be checked for leakage;

is fully supported in the Specification at: Fig. 9 step 920; Page 24 line 2 – Page 25 line 14.

The clause:

comparing a buffer consistency point number read from an old buffer with the first consistency point number, the magic numbers to identify the buffer check control structure containing the buffer consistency point number, and if the buffer consistency point number agrees with the first consistency point number, continuing to write the first consistency point, and continuing to write the new data to the new data buffers, and

in the event that the buffer consistency point number disagrees with the first consistency point number, signaling an administrator that buffer leakage between consistency points has occurred, and halting writing of the first consistency point, and halting write operations to add new data to the storage system.

is fully supported in the Specification at: Fig. 9 step 930; Page 24 line 23 – Page 25 line 14.

All independent claims are believed to be in condition for allowance.

All dependent claims are believed to be dependent from allowable independent claims.

Favorable action is respectfully solicited.

Please charge any additional fee occasioned by this paper to our Deposit Account No. 03-1237.

Respectfully submitted,

/A. Sidney Johnston/

A. Sidney Johnston
Reg. No. 29,548
CESARI AND MCKENNA, LLP
88 Black Falcon Avenue
Boston, MA 02210-2414
(617) 951-2500